# Technology Foundations for Computational Evaluation of Software Security Attributes

Gwendolyn H. Walton
Thomas A. Longstaff
Richard C. Linger

**December 2006**

**TECHNICAL REPORT**
CMU/SEI-2006-TR-021
ESC-TR-2006-021

**CERT® Program**


Carnegie Mellon

This report was prepared for the

SEI Administrative Agent
ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731-2100

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.



This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

# Table of Contents

# List of Figures

# Abstract

In the current state of practice, analysis of the security attributes of software systems is typically carried out through subjective evaluations by security experts who accumulate system knowledge in bits and pieces from architectures, specifications, designs, code, and tests. In contrast, this report describes foundations for a new computational security attributes (CSA) technology. This innovative approach provides precise computational methods for defining and analyzing security attributes based solely on the data and transformations of data found within programs. CSA permits security attributes to be evaluated through automatable analysis of the functional behavior of programs. The technology can support specification of security attributes of systems before they are built; specification and evaluation of security attributes of acquired software; verification of the as-built security attributes of systems; and real-time evaluation of security attributes during system operation.

# 1 Software Security Attributes

Fast and reliable analysis of security attributes is vital for every sector of our software-dependent society. For example, access to enterprise applications and data must be restricted to those who can provide appropriate proofs of identity. Applications and data must be protected so that attempts to corrupt them are detected and prevented. Healthcare systems must protect personal data while allowing controlled access by authorized personnel. Enterprises must be able to demonstrate that every accounting change is auditable. The flow of data through enterprise applications and the flow of transactions that drive the data must be logged and reported as proof of what actually happened.

In the current state of practice, security properties of software systems are typically assessed through labor-intensive evaluations by security experts who accumulate system knowledge in bits and pieces from architectures, specifications, designs, code, and test results. Ongoing program maintenance and evolution limit the relevance of even this hard-won but static and quickly outdated knowledge. When systems operate in threat environments, security attribute values can change very quickly. To further complicate matters, security strategies must be sufficiently dynamic to keep pace with organizational and technical change.

This report describes a fundamentally different approach to the specification and evaluation of software security attributes. This approach recognizes and leverages the fact that the problem of determining the security properties of programs comes down in large measure to the question of how the software behaves when invoked with stimuli intended to cause harmful outcomes. Because security properties have functional characteristics amenable to computational approaches, it is appropriate to focus on the question "What can be computed with respect to security attributes?" The computational security attribute approach provides a step toward a computational security engineering discipline. The ultimate goal is to develop and describe mathematical foundations and their engineering automation to permit

- rigorous specification, evaluation, and improvement of the security attributes of software and systems during development
- specification and evaluation of the security attributes of acquired software
- verification of the as-built security attributes of software systems
- real-time evaluation of security attributes during system operation

The CSA approach will support modeling, analysis, and evaluation of the security attribute values of software, as constrained by the policies of specific execution environments.

# 2 Historical Approaches to Security Attribute Analysis

The evolution of security attribute analysis is illustrated in Figure 1. Historically, security analysis has required consideration of a variety of artifacts and concerns. Early methods emphasized information and people; analysis of information attributes centered on code and models, and analysis of people attributes centered on policies. In the 1980s, the security emphasis shifted to systems analysis and architectures. By the 1990s, the emphasis shifted to survivability of the essential missions of software and systems. Survivability analyses centered on approaches to resist, recognize, recover from, and adapt to mission-compromising attacks. In addition, the 1990s yielded definitions and descriptions for security attributes, functions, actors, roles, and protocols. For examples, see those set forth by the International Standards Organization, Longstaff report, and the National Research Council [ISO/IEC 1996, Longstaff 1997, NRC 1999].

The security analyses of the past have typically been carried out through subjective evaluations by security experts. In addition, some researchers have attempted, with limited success, to apply traditional formal methods to security attributes. Researchers have applied a variety of analytical tools such as model checking, concurrent sequential process modeling, and rule-based systems to model, analyze, and verify security protocols. In recent years, several research threads have emerged that address the difficult problems of security metrics and quantitative analysis and evaluation of security attributes. This research to date has typically focused on finding approximate solutions. Many of the studies have been based on assumptions that severely constrain the operational utility of the results. In addition, many security analysis methods require that a separate model of the software be developed to include consideration of the users and/or the environment in which the software is executed. While these approaches provide valuable insights, none of them allow security analysts to state with certainty how the software under consideration behaves under ALL circumstances of use with respect to security attributes of interest.

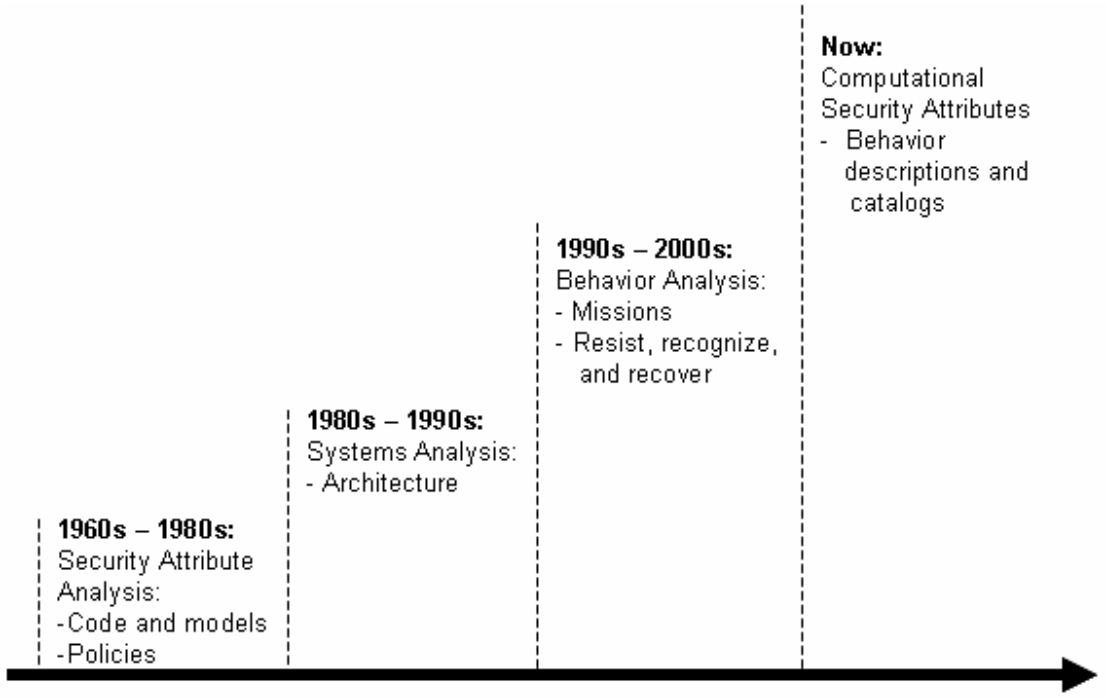


*Figure 1: Evolution of Approaches to Security Attribute Analysis*

There are many potential objects of interest to a security analyst. Both hostile and legitimate users perform data access processes, and administrators perform system security processes. Data access processes create, read, and change user and control data, identification and access privilege data of users (which must be kept confidential), and audit data. System security processes authenticate users, authorize processes, secure transmissions of data, and generate audit data. Unfortunately, the behavior of users and administrators, the operating environment, and data transmission mechanisms are all non-determinable. While there are some excellent approaches for approximating the behavior of these objects, it must be recognized that these approaches do not describe the complete behavior of the software under all circumstances of use. Software with unknown behavior cannot be certified as secure. Thus, the current technology for security analysis is inadequate. The following quotes from the National Research Council report, *Trust in Cyberspace,* underscore this point [NRC 1999]:

- "Access control policies merely model in cyberspace notions of authorization that exist in the physical world. However, in cyberspace, programs—acting on behalf of users or acting autonomously—and not the users themselves are what interact with data and access other system objects." (p111)
- "Trustworthiness mechanisms basically concern events that are not supposed to happen... some users may be malicious, and the world is not fault free." (p63)
- "Some properties (e.g., 'the absence of security vulnerabilities') have no system-independent formalization and, therefore, are not amenable to direct analysis using formal methods." (p96)

- “For any given system, there will exist properties that together imply ‘the absence of security vulnerabilities.’ But careful thought by a system developer is required to identify these constituents, and there is no formal way to ever establish that the system developer has listed them all.” (p96)
- “An overwhelming majority of security vulnerabilities are caused by buggy code.” (p110)

A method is needed for security attribute specification and analysis based on 1) the actual behavior of software, and 2) security attribute definitions that consider only data and transformations on data. The computational security attributes analysis approach (shown under the “Now” heading in the timeline of Figure 1) emphasizes the comparison of behavioral descriptions of security attributes with behavior catalogs. These catalogs completely describe the behavior of the code and are produced by the new technology of function extraction (FX).

# 3 Security Attribute Definitions

Published definitions for security attributes lack theoretical foundations and cannot support a disciplined approach to analysis of software and system security. For example, various U.S. government publications describe *privacy* as an interest, a freedom, an ability, and a right:

- For purposes of the Health Insurance Portability and Accountability Act (HIPAA) Privacy Rule, privacy means "an individual's interest in limiting who has access to personal health care information" [US OCR 2006].
- The NRC report entitled *Trust in Cyberspace* describes privacy as "freedom from unauthorized intrusion" [NRC 1999].
- The NSA 'red book' defines privacy as follows: "(1) the ability of an individual or organization to control the collection, storage, sharing, and dissemination of personal and organizational information. (2) The right to insist on adequate security of, and to define authorized users of, information or systems" [NSA 1987].

The National Security Agency (NSA) 'red book' definition further states: "The concept of privacy cannot be very precise and its use should be avoided in specifications except as a means to require security, because privacy relates to 'rights' that depend on legislation" [NSA 1987].

In contrast, U.S. government publications describe *authorization* both as a process and as a document:

- With respect to HIPAA, authorization refers to the document that designates permission [US OCR 2006].
- For the purpose of the Gramm-Leach-Bliley Act, authorization is the "process by which a known (not anonymous) entity gains specified privileges such as access, read or write rights, system administration rights, etc." [FDIC 2004].

Published definitions can be more or less restrictive, can require separate security policies for different organizations or geographical areas, and can leave much room for interpretation. For example, consider the following U.S. government definitions for *confidentiality*:

- Confidentiality is "the protection of communications traffic or stored data against interception or receipt by unauthorized third parties" [NRC 1999].
- The meaning of confidentiality may vary across states, and even across companies: "the protection of individually identifiable information as required by state and federal legal requirements and Partners policies" [US OCS 2006].

- Privacy can also be interpreted as “against unauthorized access to or use of customer information that could result in substantial harm or inconvenience to any customer” [FDIC 2004].

To support a behavioral approach to security attribute specification and analysis, it is necessary to develop consistent definitions that can support computational analysis of security attributes. The following descriptions of security attributes provide useful information to support developing such definitions [ISO/IEC 1996, Longstaff 1997, NRC 1999]:

- Confidentiality: “the protection of communications traffic or stored data against interception or receipt by unauthorized third parties” [NRC 1999]. “When information is read or copied by someone not authorized to do so, the result is known as loss of confidentiality” [Longstaff 1997]. “The confidentiality function prevents the unauthorized disclosure of information. The confidentiality function includes the functions hide and reveal. In the context of confidentiality, objects fulfill either or both of the following roles: confidentiality-protected information originator or confidentiality-protected information recipient” [ISO/IEC 1996].
- Integrity: “the property of an object meeting an a priori established set of expectations. In the distributed system or communication security context, integrity is more precisely defined as assurance that data have not been undetectably modified in transit or storage” [NRC 1999]. “When information is modified in unexpected ways, the result is known as loss of integrity. This means that unauthorized changes are made to information, whether by human error or intentional tampering” [Longstaff 1997]. “The integrity function detects and/or prevents the unauthorized creation, alternation, or deletion of data. The integrity function includes all the following functions: shield, validate, unshield. In the context of integrity, objects fulfill one or more of the following roles: Integrity-protected data originator or integrity-protected data recipient” [ISO/IEC 1996].
- Availability: “the property asserting that a resource is usable or operational during a given time period, despite attacks or failures” [NRC 1999]. “Information can be erased or become inaccessible, resulting in loss of availability. This means that people who are authorized to get information cannot get what they need. When a user cannot get access to the network or specific services provided on the network, they experience a denial of service” [Longstaff 1997].
- Authentication: “proving that a user is who he or she claims to be. That proof may involve something the user knows (such as a password), something the user has (such as a smartcard), or something about the user that proves the person's identity (such as a fingerprint)” [Longstaff 1997]. “Authentication refers to the process by which a system establishes that an identification assertion is valid” [NRC 1999]. “In the context of authentication, objects fulfill one or more of the following roles: principal, claimant, or trusted third party” [ISO/IEC 1996]. Authentication requires use of exchange authentication information. Authentication is of interest when communication is required in the (possible) presence of a hostile agent.

- Authorization: "the act of determining whether a particular user (or computer system) has the right to carry out a certain activity, such as reading a file or running a program" [Longstaff 1997].
- Non-repudiation: when the means of authentication cannot later be refuted—the user cannot later deny that he or she performed the activity. "The non-repudiation function prevents the denial by one object involved in an interaction of having participated in all or part of the interaction. In the context of non-repudiation, objects fulfill one or more of the following roles: (non-repudiable data) originator, (non-repudiable data) recipient, evidence generator, evidence user, evidence verifier, non-repudiation service requester, notary, or adjudicator" [ISO/IEC 1996].
- Confidentiality: "the protection of communications traffic or stored data against interception or receipt by unauthorized third parties." [NRC 1999] "When information is read or copied by someone not authorized to do so, the result is known as loss of confidentiality." [Longstaff 1997] "The confidentiality function prevents the unauthorized disclosure of information. The confidentiality function includes the functions hide and reveal. In the context of confidentiality, objects fulfill either or both of the following roles: confidentiality-protected information originator, or confidentiality-protected information recipient" [ISO/IEC 1996].
- Integrity: "the property of an object meeting an a priori established set of expectations. In the distributed system or communication security context, integrity is more precisely defined as assurance that data have not been undetectably modified in transit or storage" [NRC 1999]. "When information is modified in unexpected ways, the result is known as loss of integrity. This means that unauthorized changes are made to information, whether by human error or intentional tampering" [Longstaff 1997]. "The integrity function detects and/or prevents the unauthorized creation, alternation, or deletion of data. The integrity function includes all the following functions: shield, validate, unshield. In the context of integrity, objects fulfill one or more of the following roles: Integrity-protected data originator or integrity-protected data recipient" [ISO/IEC 1996].
- Availability: "the property asserting that a resource is usable or operational during a given time period, despite attacks or failures" [NRC 1999]. "Information can be erased or become inaccessible, resulting in loss of availability. This means that people who are authorized to get information cannot get what they need. When a user cannot get access to the network or specific services provided on the network, they experience a denial of service" [Longstaff 1997].
- Authentication: "proving that a user is whom he or she claims to be. That proof may involve something the user knows (such as a password), something the user has (such as a smartcard), or something about the user that proves the person's identity (such as a fingerprint)" [Longstaff 1997]. "Authentication refers to the process by which a system establishes that an identification assertion is valid" [NRC 1999]. "In the context of authentication, objects fulfill one or more of the following roles: principal, claimant, or trusted third party" [ISO/IEC 1996]. Authentication requires the use of exchange

authentication information. Authentication is of interest when communication is required in the (possible) presence of a hostile agent.

- Authorization: "the act of determining whether a particular user (or computer system) has the right to carry out a certain activity, such as reading a file or running a program" [Longstaff 1997].
- Non-repudiation: when the means of authentication cannot later be refuted - the user cannot later deny that he or she performed the activity. "The non-repudiation function prevents the denial by one object involved in an interaction of having participated in all or part of the interaction. In the context of non-repudiation, objects fulfill one or more of the following roles: (non-repudiable data) originator, (non-repudiable data) recipient, evidence generator, evidence user, evidence verifier, non-repudiation service requester, notary, or adjudicator" [ISO/IEC 1996].

# 4 Function Extraction Technology

The mathematics of functions provides a solid point of departure for computational analysis of security attributes. Desired security attributes can themselves be specified in terms of functions (i.e., in terms of data and transformation on data), thereby permitting software to be evaluated for conformance or not through comparison of behavioral requirements of security attributes to the functional behavior of the software. The emergence of CERT's new function extraction (FX) technology, unavailable to previous researchers, provides the critical first step for computational security attribute analysis by supporting the derivation of the full functional behavior of programs as a starting point for security analysis.

To see how FX works, consider the miniature Intel assembly language program of Figure 2 and the question of what it does. The figure shows the relative address (in hex) of each line of code after it has been disassembled. Note that t1, t2, and t3 are labels used by jump (jmp) instructions to branch to the line of code at that address. The arrows show these branches to reveal the spaghetti-logic control flow of the code.

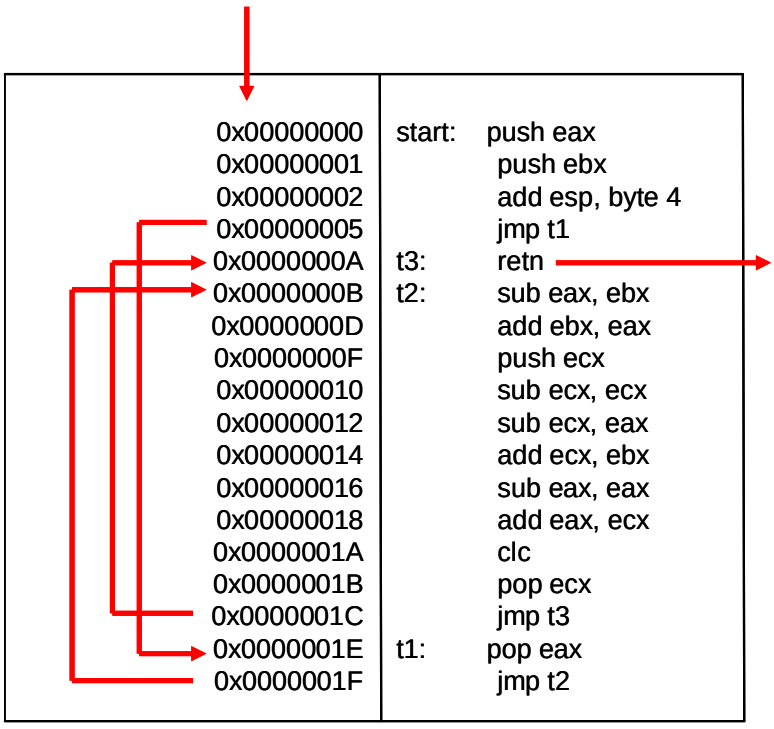


*Figure 2: An Assembly Language with Three Jumps*

In terms of their behavior, assembly language programs can do only three things: update registers, update flags, and update memory. It is certainly possible to determine exactly what the program of Figure 2 does by manually performing a step-by-step analysis of the semantics of each instruction in the program. However, this task is time consuming and potentially error prone. A better approach is to extract the function computationally using FX technology. The first step in FX processing is to transform the program into structured form expressed in terms of the three fundamental control structures of sequential logic: sequences, alternations (if-then-else), and iterations (loops). The next step is to extract the behavior of the structured version by using a precise definition of the functional semantics of each instruction in the language of interest (Intel assembly language in this example) together with rules for their combination. The result is a complete, correct derivation of the behavior of the code. This definition reveals the net behavior of the program, that is, how it transforms input data into output data, essentially its as-built specification.

To illustrate, Figure 3 shows a screen image of output from the first generation of a prototype system (Function Extraction for Malicious Code - FX/MC) under development by CERT STAR*Lab to support function extraction of assembly code.

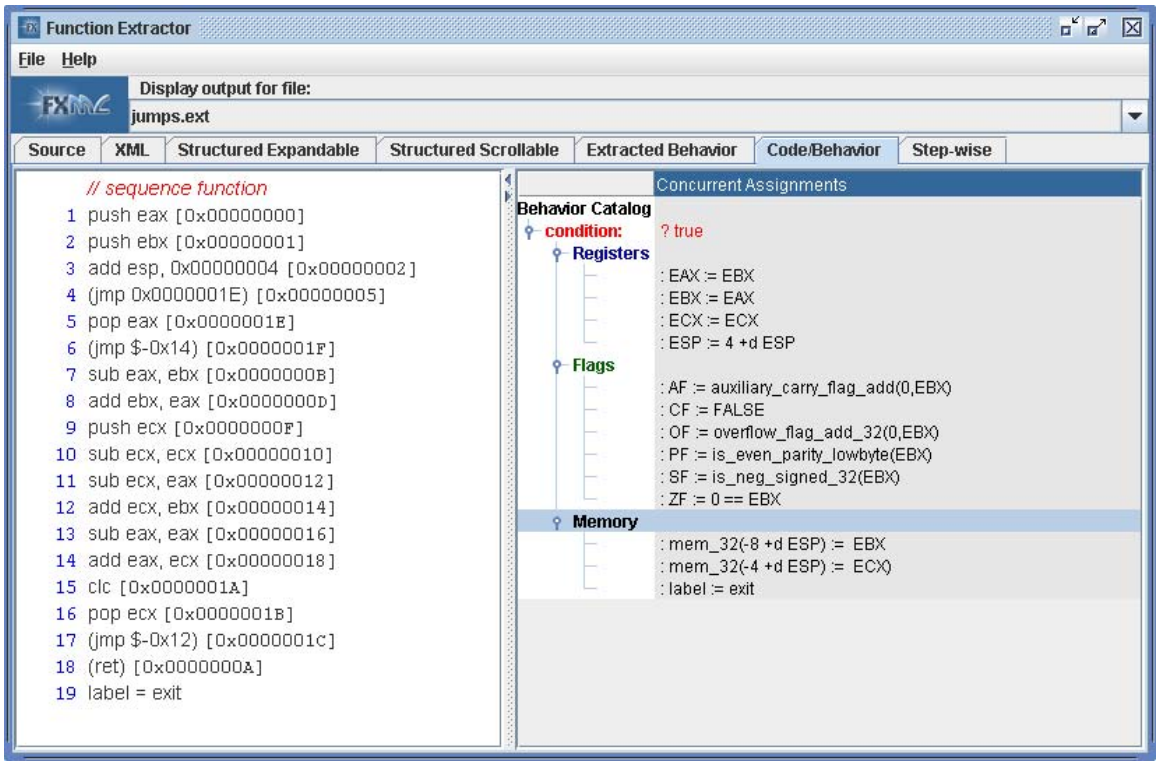


*Figure 3: FX/MC Output after Processing the Program of Figure 2*

The window on the left side of Figure 3 shows the code of Figure 2 after it has been structured. The information enclosed in parentheses and square brackets in the structured output are comments to assist the reader in comparing the structured version with the original. The original address for each line of code is shown in square brackets at the end of the line. The jumps are no longer relevant, since after structuring they simply jump to the next line of code. Jumps are included in the structured listing in the left window as comments to assist the reader who wishes to compare the structured code with the original code. Note that the only purpose of the jumps in this example seems to have been to obfuscate the code, because after the FX system unraveled the spaghetti-logic of Figure 2, only a simple sequence of instructions remained.

The window on the right-hand side of Figure 3 shows the complete behavior of the code as computed by the system. The behavior description is in the form of non-procedural conditional concurrent assignments that show the behavior in terms of transformations on the input data. The first line of the behavior catalog, *condition: ?true*, indicates that there is no condition controlling this code's behavior. Thus, the program of Figure 2 always behaves in the same way under all circumstances, regardless of the initial state of the registers, flags, and memory.

The concurrent assignments in the behavior catalog are displayed into three sections: register updates, flag updates, and memory updates. The left-hand side of each concurrent assignment represents a final value; the right-hand side represents how the final value is computed from initial values. Thus, the behavior catalog shows that only the three general-purpose registers (*EAX*, *EBX*, and *ECX*) and the stack pointer (*ESP)* are required to express the behavior of the code. According to the right-hand window of Figure 3, the calculated behavior can be summarized as follows:

- Registers: The final value stored in *ECX* was the same as the initial value, but the code will swap the values of *EAX* and EBX. The value of the stack pointer will be increased by 4.
- Two values will remain in memory on the stack after the code is executed: *EBX* and *ECX*.
- The values of six flags will be changed (*AF, CF, OF, PF, SF,* and *ZF*).

This behavior might not be obvious from a casual reading of either the original code or even the structured version. Looking only at the *push* and *pop* instructions of the structured code in the left-hand window of Figure 3, one would note the following sequence.

```
push   eax
push   ebx
pop    eax
push   ecx
pop    ecx
```

If this sequence of push and pop instructions were considered in isolation from the rest of the code, it would appear that the final value for EAX will be the initial value of EBX, and one

value (the initial value of EAX) will remain on the stack. However, this simplistic analysis gives an incorrect result for both register assignments and memory values due to the add instruction that manipulated the stack pointer (see line 3 of the structured code: *add esp, 0x00000004*), and does not consider any changes to the flags. Certainly, for this miniature example, a careful programmer with knowledge of the Intel instruction set semantics could manually do the thorough analysis required to derive the complete, correct behavior provided in the behavior catalog. However, for large, complex programs, such a labor-intensive, error-prone manual analysis is simply not feasible.

FX technology can be applied to complex programs to yield complete calculated behavior expressed in terms of net effects on data. For example, Figure 4 shows the output produced by the FX system in computing the complete behavior of a 4000-line assembly language program that included many superfluous statements to obfuscate the true behavior of the code. As shown in the right-hand window of Figure 4, these 4000 lines of code do nothing other than store a value of 4 in the general-purpose EAX register, change the value of the stack pointer, and set six flags. Such intentional obfuscation is often found in malicious code.

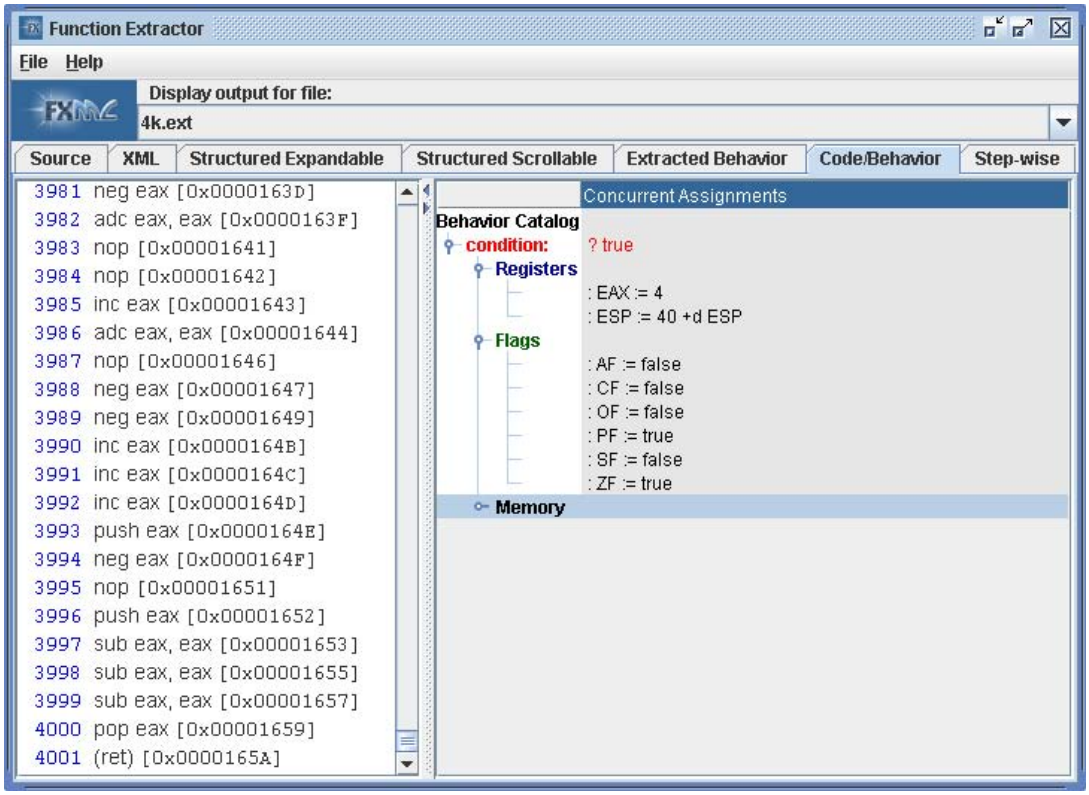


*Figure 4: FX/MC Behavior Catalog for a 4000-line Assembly Language Program*

FX technology can be applied to any programming language environment and has the potential to impact many aspects of the software engineering lifecycle. The function extraction process derives the as-built specification of software; that is, the behavior that has actually been implemented. This derived behavior can be compared to requirements and

specifications to determine if the software is indeed a correct implementation. Thus, the derived behavior can be used to determine if the software meets security requirements if they have been specified in behavioral terms.

FX technology prescribes effective means to create and record specifications, with the corresponding specification task itself amenable to automated support. Automated correctness verification of code with respect to desired security properties would be especially valuable during system development, to check on the behavior of partial implementations and find and fix errors and vulnerabilities along the way. It would also permit a new level of rigor in acquisition and acceptance of systems by supporting required provision of behavior catalogs for all delivered code.

# 5 The CSA Analysis Process

While analysts have often characterized many security attributes as "non-functional" properties of programs, it turns out that they are in fact fully functional and thereby subject to FX-style automated analysis. Complete definitions of the required behavior of security attributes of interest can be created based solely on data and transformations of data. These definitions can then be used to analyze the security properties of programs. Thus, as illustrated in Figure 5, computational security attribute analysis consists of three steps:

1. **Define required security behavior.** Specify security attributes in terms of required behavior during execution expressed in terms of data and transformations on data.
2. **Calculate program behavior.** Apply function extraction to create a behavior catalog that specifies the complete "as built" functional behavior of the code.
3. **Compare program behavior to required security behavior**. Compare the computed behavior catalog with required security attribute behavior to verify whether it is correct or not.

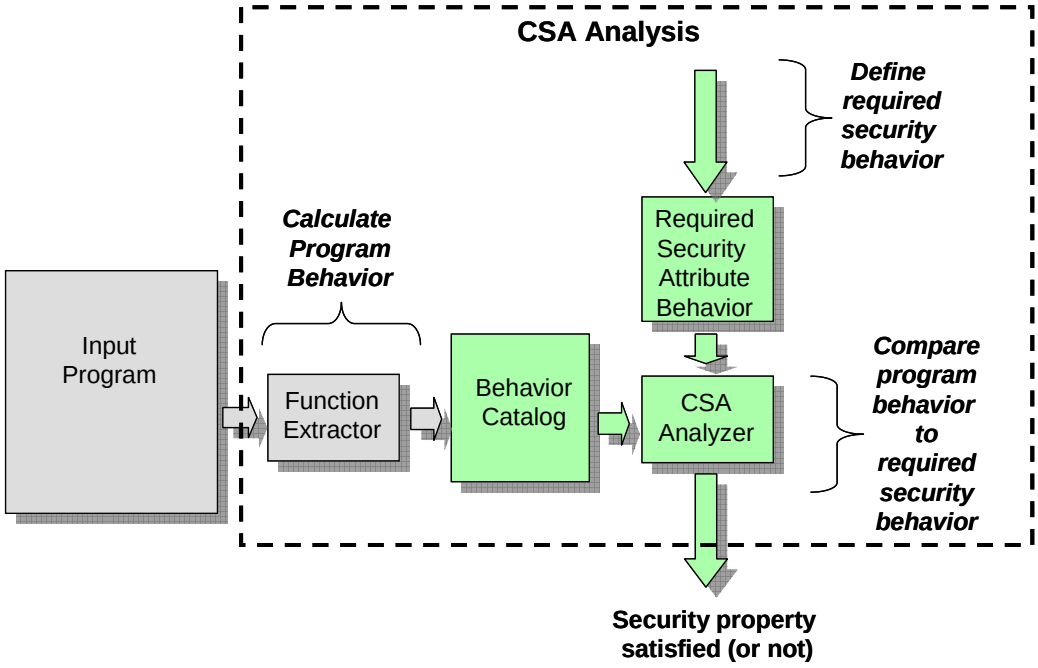


*Figure 5: The CSA Approach*

Requirements for security attribute behavior must explicitly define expected behavior of code in all circumstances of interest. Thus, the requirements for security attribute behavior must include a minimal definition of required behavior for all inputs of interest to the security attributes, including desired inputs (for example, an authenticated user id) and undesired inputs (for example, an unknown user id). Usage environment conditions related to security attributes are specified in the same manner as inputs to the system. For example, availability of the network might be specified by a Boolean value that indicates whether or not the network is currently available. Security successes and failures are also specified in terms of data. For example, system control data can be used to indicate whether the current user has been authenticated using a trusted authentication mechanism.

The level of abstraction at which a security attribute is specified can depend on the specific situation. For example, if all available data transmission mechanisms have previously been certified to be trusted, the security attribute requirements would need not include details regarding data transmission. If there is one trusted data transmission mechanism, X, and one or more data transmission mechanisms that may not be trustworthy, the security attribute requirements could specify that all data transmissions will be performed using X. If none of the data transmission mechanisms have been previously certified as trusted, the security attribute requirements will need to include required control data effects for transmission security.

A “never responded” and “no output” case for each external function call of interest must be considered, including a definition of correct behavior in the case of intentional and unintentional aborts and hangs. In addition, security attribute requirements may specify a specific order in which certain functions can be called. For example, user authentication must occur before any data access functions can be called. The requirements may specify that a certain set of data transformations always occur. For example, the control data that indicates that data transmission is secure must always be set by the trusted data transmission mechanism. The requirements may specify that a certain set of data transformations must never occur. For example, the control data that indicates that data transmission is secure must never be set by any code other than the trusted data transmission mechanism. In the case where a user is authorized to only access specific data, the requirements may state that no data transformations other than those specified can occur.

Any amount of traceability and control can be specified in the requirements for security attribute behavior. For example, the requirements may include specifications of bounded behavior. (i.e., execution will proceed so long as the behavior is within a specified domain) Specifications for trusted mechanisms can be included in the requirements as constraints. For example, one might specify that “a call to method XXX that returns a value of *y* is sufficient to satisfy a requirement that a trusted mechanism must be used to perform authentication.” The behavioral approach also supports dealing with some uncertainty in the specification of the security attribute requirements. For example, a security requirement might state that the code must guarantee security properties modulo some defined value. Some constraints might

be specified using a stochastic component. For example, “The response history of component X must indicate that the component was available at least 94% of the time.”

Verification that a security property is satisfied requires verification of both the data at rest (i.e., the control data values) and the data in motion (i.e., the mechanisms used to perform the data transformations). Some common tasks to verify data at rest include checking to make sure that a specific task (for example, an audit task) will always be carried out to validate the contents of a specific control data structure. Advantages of this approach to security attribute verification include the use of constraints and boundary conditions that can make any assumptions explicit. People and process issues can be handled by the CSA approach by using assumptions and constraints as part of the behavior catalogs. Behaviors can embody requirements for a given security architecture. The attribute verification process will expose security vulnerabilities, making it easier to address evolution of code, environment, use, and so forth.

The CSA verification process can provide important opportunities for improved acquisition and third-party verification. A “user” of a system might be a person, a device, or a software component. The user may be the intended user or may be an unexpected and/or hostile user. An issue that must be considered with commercial off-the-shelf (COTS) products and reuse is that the definition of “user” embodied in the security behavior requirements may not be the same definition that was employed in the COTS or reused component. The same issue occurs when unknown components are employed as “black boxes” in systems of systems. If, in the composition of components or systems, it doesn’t matter what a specific “black box” component does with respect to security attribute requirements, then that component can be used. However, if the behavior of a component does matter, it cannot be used until its security attributes have been verified. In this case, a behavior catalog can be calculated for the component using its executable, even if documentation and source code are not available. Only externally observable behaviors are of interest to security attribute analysis. Thus, while the behavior catalog will have to be produced for the entire system in order to extract the externally observable behaviors, there is no need to expose the algorithm or source code, and there’s no need to understand the entire state space.

# 6 Behavioral Requirements of Security Attributes

As noted earlier, security attributes are often termed “non-functional” properties. In reality, security properties are fully functional and are dependent on the execution behavior of software. The security attributes discussed in this report are (listed alphabetically) availability, authentication, authorization, confidentiality, integrity, non-repudiation, and privacy. Three of these attributes (confidentiality, integrity, and availability) are important to information. The other four attributes (authentication, authorization, non-repudiation, and privacy) relate to the people who use that information.

The behavioral requirements for each of these attributes can be completely described in terms of data items and constraints on their processing. The processing can be expressed, for example, as logical or quantified expressions or even conditional concurrent assignments, which can be mechanically checked against the calculated behavior of the software of interest for conformance or non-conformance with security attribute requirements.

## 6.1 Use of a Trusted Mechanism

Each of the security attributes requires the use of one or more trusted mechanisms. FX technology can be used to certify a mechanism as trusted. A behavior catalog for a mechanism can be generated to describe all cases of behavior in terms of its data and the transformations it carries out on that data. The behavior catalog can then be analyzed to ensure the following:

- The trusted mechanism sets the values of control data which indicates whether the mechanism executed correctly.
- If control data indicates that the mechanism executed correctly, there exists evidence data to show that the data transformation was performed in a manner that satisfies the defined security specification.

Note that the implementation of a security attribute may include a trusted third party to acquire, authenticate, and adjudicate evidence of transactions. However, for the purposes of behavior specification of security attributes, the specific mechanism and actors are not relevant. All that is needed is a precise specification of the data and the transformations on the data, and any constraints concerning these transformations.

## 6.2 Trusted Data Transmission

As illustrated in Figure 6, the requirements for trusted data transmission are as follows:

- A trusted data transmission mechanism is used for all data transmissions. If the mechanism is not available or the mechanism fails, the requirement fails.
- No mechanism outside this trusted data transmission mechanism sets the value of the control data that indicates whether the data transmission mechanism executed correctly.

As illustrated in Figure 7, the process for determining whether data transmission security properties are satisfied by the data transmission components of a system consists of the application of the CSA analysis process illustrated in Figure 5. The input to the process is the data transmission components of the system, and the output is a determination of whether the data transmission security requirements have been satisfied.

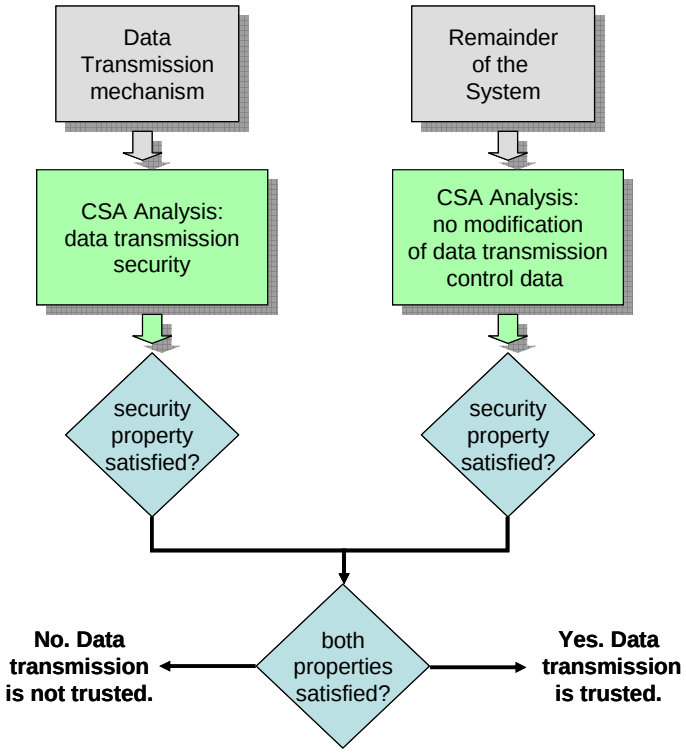


*Figure 6: Requirements for Trusted Data Transmission*

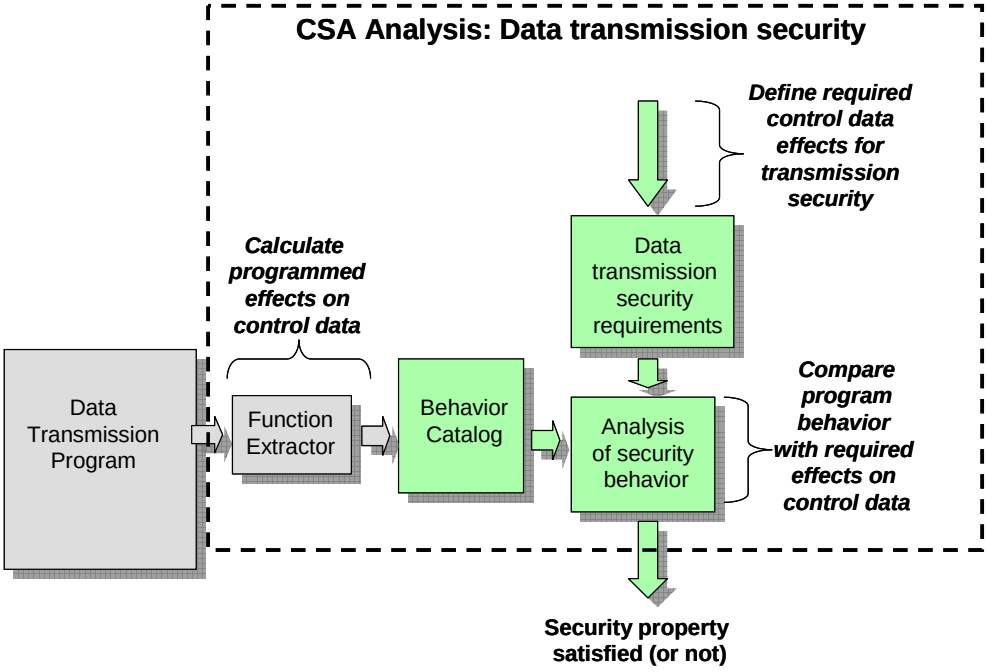


*Figure 7: CSA Analysis of Data Transmission Security*

As illustrated in Figure 8, the process for determining whether the remainder of the system can modify the control data set by the data transmission components consists of the application of the CSA analysis process, where the input to the process is the software for the entire system, and the output is a determination of transmission components can be modified by any other part of the system. whether the control data set by the data

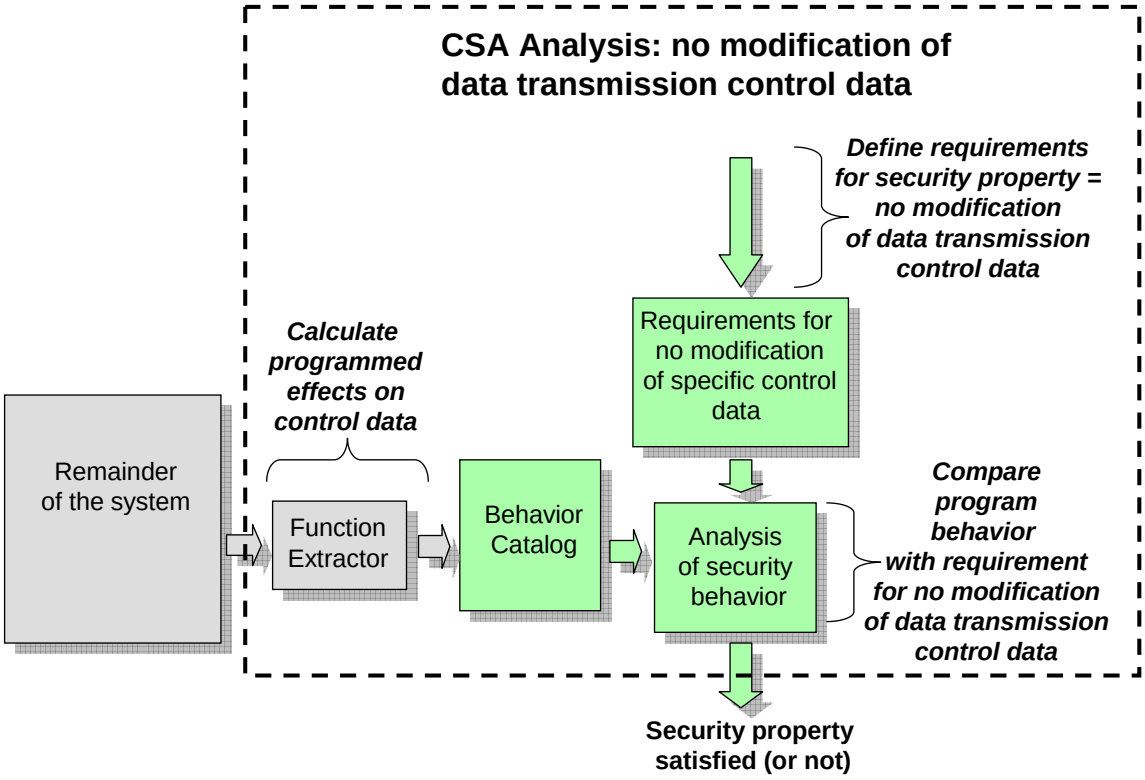


*Figure 8: CSA Analysis of Modification of Data Transmission Control Data*

To verify that a data transmission mechanism is trusted, one must verify the data that provides the evidence related to data transmission. For example, the specification for the data to provide evidence of valid data transmission might describe the mechanism by which each data message output incorporates a shared (between sender and receiver) data item that can be used to verify that the transformation worked correctly. Assignments to this shared data must not be reversible (i.e., guaranteed encryption).

As another example, suppose the FX behavior catalogs for all of the code have been examined to verify that all data transmissions in the system occur as a result of calls to function YYY. To verify the necessary security properties for data transmission, we examine function YYY's behavior catalog to determine the net effect of the data transformations related to any conditions for which invalid data transmission could occur. If invalid data transmission can occur when, say, the value of register EAX is equal to 4 at a particular line of code, we calculate the behavior catalog for all of the code up to that line and examine the resulting conditional current assignments to see the conditions (if any) for which the net effect of the data transformations is to set EAX to 4. (If the line of code of interest is in the middle of a loop, the behavior catalog will provide the values for the variables in the loop body at each step through the loop.) As still another example, if function YYY is called using an argument popped from the top of the stack, we must examine the behavior catalog of the calling program to determine the net effect of the data transformations on the stack prior to that function call to determine the value of the parameter.

## 6.3 The Authentication Security Attribute

*Authentication requires that a trusted user has been bound to the behavior*. That is, the system will only allow the program to be executed if the user has previously been determined to be a trusted user. To verify authentication, one must examine the net effects on the control data related to authentication: verify the data that provides evidence that the binding took place, and verify that this evidence data was not changed before completion of any operation that required authentication. As illustrated in Figure 9, the requirements for authentication are as follows:

- A trusted data transmission mechanism is always used for every data transmission.
- A trusted identification mechanism is always used to provide proof of a user's identification. Note that the "user" to be identified may be a person, a process, a program, or other entity.
- A trusted binding mechanism is always used to bind user data (user identification, password, or other information to confirm the identification, and the system information that provides the proof of the identification) to an execution environment.
- No other mechanism outside the trusted mechanisms sets the value of any of the control data that indicates whether each of the trusted mechanisms executed correctly and that indicates the status of the bound data.
- If any of the above requirements or mechanisms fails, authentication fails.

***Error! Objects cannot be created from editing field codes.***

*Figure 9: Requirements for Authentication Property*

The mechanism for using CSA to determine whether the data transmission is trusted was illustrated earlier (see Figures 6, 7, and 8). As illustrated in Figures 10 and 11, the analysis of the user identification mechanism and the user binding mechanism applies the CSA approach by proceeding along the same lines as Figures 7 and 8 to determine whether each of these mechanisms is trusted.

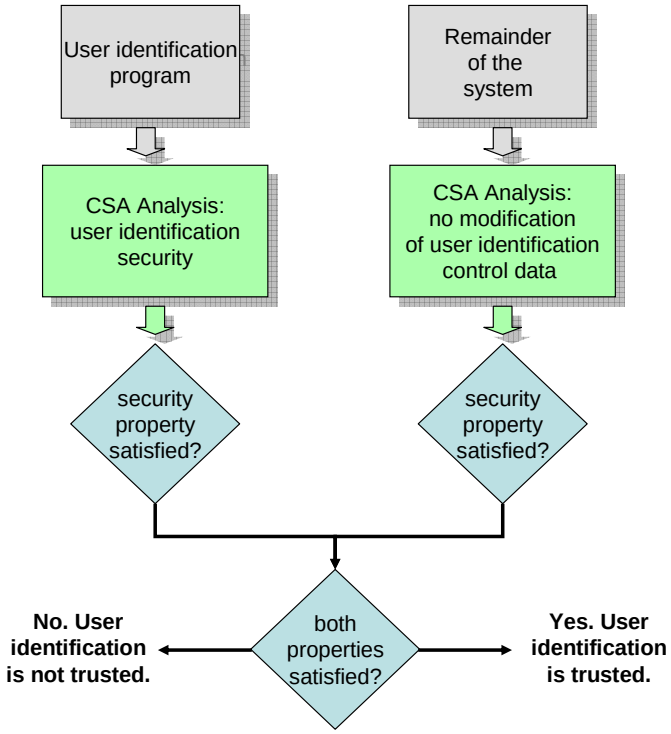


*Figure 10: CSA Analysis of Trusted User Identification*

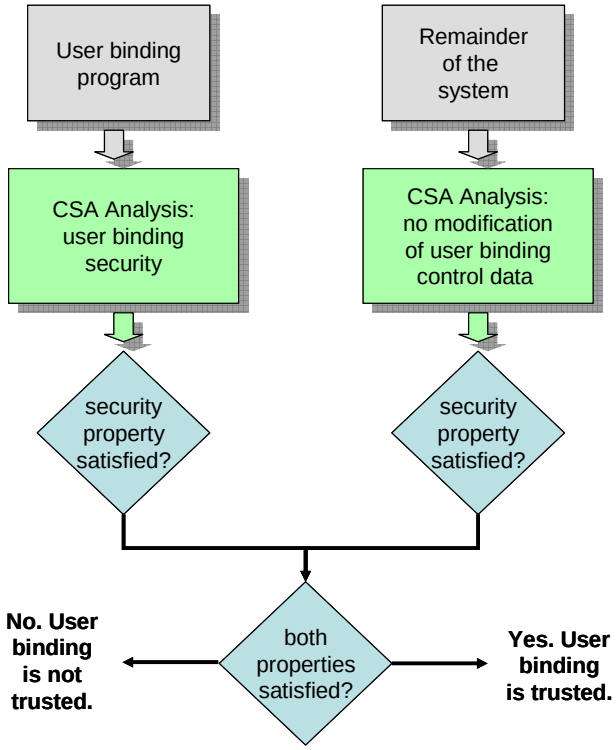


*Figure 11: CSA Analysis of Trusted User Binding*

## 6.4 The Authorization Security Attribute

*Authorization requires that a user has the right to perform the requested process.* To verify that an authorized operation took place, one must examine the net effects on the control data to verify that it provides evidence that authorization occurred before the operation, and that the evidence data for the authorization was not changed before that operation completed. As illustrated in Figure 12, the requirements for authorization are as follows:

- A trusted authentication mechanism (subsection 6.3) is always used to authenticate the user. Note that this requirement includes a requirement for trusted data transmission and authentication.
- A trusted lookup mechanism is always used to determine that the user has the right to complete the specified request.
- No other mechanism outside the trusted mechanisms sets the value of any of the control data that indicates whether each of the trusted mechanisms executed correctly and that identifies the authorized user and the scope of the authorization.
- If any of the above requirements or mechanisms fails, authentication fails.

Analysis of the authentication mechanism was discussed in the previous subsection. Analysis of the lookup mechanism applies the CSA approach by proceeding along the same lines as Figures 7 and 8 to determine whether this mechanism is trusted.

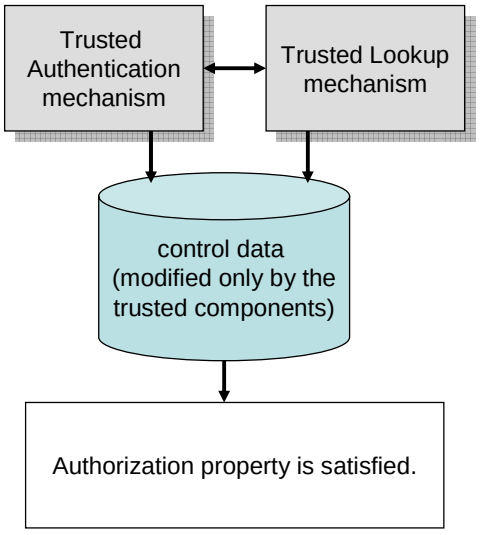


*Figure 12: Requirements for Authorization Property*

## 6.5 The Non-repudiation Security Attribute

*Non-repudiation of data transmission requires that neither the sender nor the recipient of the data can later refute his or her participation in the transaction. Non-repudiation of changes to a dataset requires that the means for authentication of changes cannot later be refuted.* For the purposes of this discussion we treat data change as a special case of data transmission, where receipt of the data transmission includes making and logging the requested change to the dataset. To verify non-repudiation one must examine the net effects on the control data related to non-repudiation. As illustrated in Figure 13, the requirements for every data transmission that is subject to non-repudiation are as follows:

- Trusted authorization (subsection 6.4) is always used for sender, receiver, and the scope of any data changed or transmitted. Note that this requirement includes a requirement for trusted data transmission, trusted authentication of users, and trusted authorization of users and processes for the specific data scope.
- Trusted binding is used to bind the sender to the data sent and to bind the receiver to data received.
- The authorization, binding, and data transmission are handled as a single atomic operation within the boundary of the authorized secure process.
- A trusted mechanism is always used to provide traceability and audit. This trusted mechanism ensures data persistence of the audit data so the means of authentication and the data transmission cannot later be refuted.
- Every data transmission is preceded by an absolute definition of the data and identification that binds the data to the sender.
- Every data receipt is preceded by an absolute definition of the data and identification that binds the data to the recipient.
- No other mechanism outside the trusted mechanisms sets the value of any of the control data that indicates whether each of the trusted mechanisms executed correctly or the value of any of the control data generated by the trusted mechanisms.
- If any of the above requirements or mechanisms fails, non-repudiation fails.

Analysis of the authorization mechanism was discussed in the previous subsection. Analysis of the binding mechanism and the traceability and audit mechanism applies the CSA approach by proceeding along the same lines as Figures 7 and 8 to determine whether each of these mechanisms is trusted.

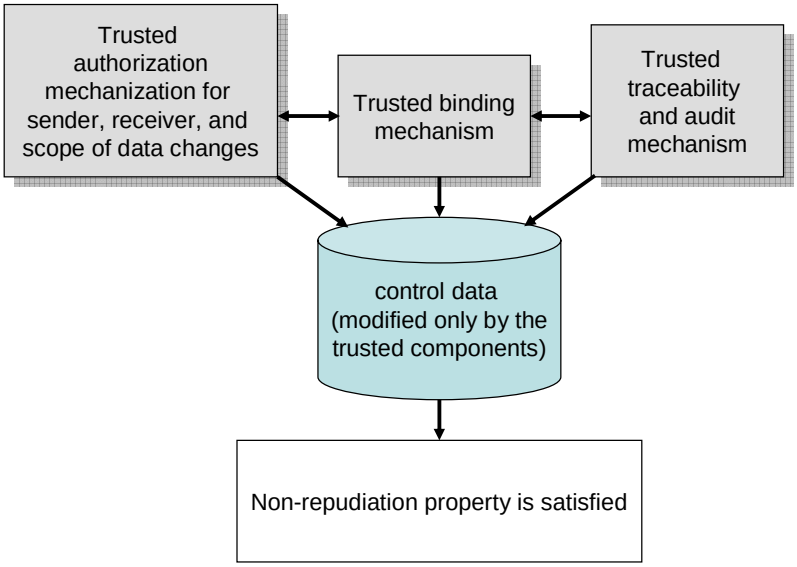


*Figure 13: Requirements for Non-repudiation Property*

## 6.6 The Confidentiality Security Attribute

*Confidential data access or confidential data transmission requires that unauthorized disclosure of one or more specific data items will not occur.* Confidentiality is often described in terms of a security policy that specifies the required strength of the mechanisms that ensure that the data cannot be accessed outside the system. For example, the security policy may require verification that approved encryption mechanisms are used for the output. To verify confidentiality, one must examine the net effects on the control data related to confidentiality. As illustrated in Figure 14, the requirements for confidentiality are as follows:

- A trusted non-repudiation mechanism (subsection 6.5) is always used to process requests for confidential data access and confidential data transmission. Note that this requirement includes a requirement for trusted data transmission, trusted authentication of users and processes, and trusted authorization of users and processes for the particular data scope.
- A trusted mechanism is always used to ensure that the data cannot be read outside the system.
- No other mechanism outside the trusted mechanisms sets the value of any of the control data that indicates whether each of the trusted mechanisms executed correctly or the value of any of the control data set by the trusted mechanisms.
- If any of the above requirements or mechanisms fails, the request for confidential data access or confidential data transmission fails.

Analysis of the non-repudiation mechanism was discussed in the previous subsection. Analysis of the data access mechanism applies the CSA approach by proceeding along the same lines as Figures 7 and 8 to determine whether this mechanism is trusted.

**Error! Objects cannot be created from editing field codes.**

*Figure 14: Requirements for Confidentiality Property*

## 6.7 The Privacy Security Attribute

*Privacy requires that an individual has defined control over how his/her information will be disclosed.* To verify privacy, one must examine the net effects on the control data related to privacy. As illustrated in Figure 15, the requirements for privacy are as follows:

- A trusted confidentiality mechanism (subsection 6.6) is always used for all accesses of a user's personal information. Note that this requirement includes a requirement for trusted data transmission, trusted authentication of users and processes, trusted authorization of users and processes for the specific data scope, and trusted non-repudiation for access to a user's personal information.
- All access to a user's personal information satisfies an existing privacy and confidentiality policy that includes control data that defines the scope of access for each user.
- A trusted non-repudiation mechanism (subsection 6.4) is used for all changes to the control data that defines the scope of access for each user. Note that this requirement includes a requirement for trusted data transmission, trusted authentication of users, and processes and scope of data.
- No other mechanism outside the trusted mechanisms sets the value of any of the control data that indicates whether each of the trusted mechanisms executed correctly or the value of any data set by the trusted mechanisms.
- If any of the above requirements or mechanisms fails, the request for confidential data access or confidential data transmission fails.

Analysis of the confidentiality and non-repudiation mechanisms was discussed in previous subsections. Analysis of the access mechanism applies the CSA approach by proceeding along the same lines as Figures 7 and 8 to determine whether this mechanism is trusted.

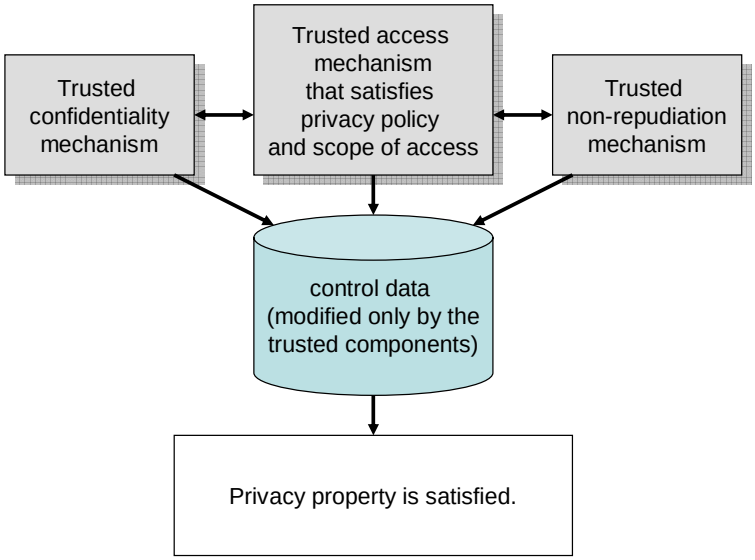


*Figure 15: Requirements for Privacy Property*

## 6.8 The Integrity Security Attribute

*Integrity requires that authorized changes are allowed, changes must be detected and tracked, and changes must be limited to a specific scope*. Integrity is defined as a property of an object, not of a mission. To verify integrity, one must examine the net effects on the control data related to integrity. That is, one must be able to: isolate the object, isolate all the behaviors that can modify the object, detect any modifications to the data, and ensure that all transformations of the data across the object are within the pre-defined allowable subset. As illustrated in Figure 16, the requirements for integrity are as follows:

- A security policy exists that describes the scope of allowed changes as an invariance function: certain data transformations must hold; others must never hold.
- If the security policy data is changed to remove any element from the allowable subset, integrity of the data fails.
- A trusted non-repudiation mechanism (subsection 6.5) is always used for changes to data and changes to policy to ensure that all changes to the security policy and changes to data are performed using a trusted non-repudiation mechanism. Note that this requirement includes a requirement for trusted data transmission, trusted authentication of users and processes, trusted authorization of users and processes for the specific data scope, and trusted non-repudiation for changes to the data. Every authorization for data changes must be restricted to the allowable subset as defined in the security policy.
- No other mechanism outside the trusted mechanisms sets the value of any of the control data that indicates whether each of the trusted mechanisms executed correctly or the value of the control data set by any of the trusted mechanisms.
- If any of the above requirements or mechanisms fails, integrity of the data fails.

Analysis of the non-repudiation mechanism was discussed in a previous subsection. Analysis of the data change mechanism applies the CSA approach by proceeding along the same lines as Figures 7 and 8 to determine whether this mechanism is trusted.

**Error! Objects cannot be created from editing field codes.**

*Figure 16: Requirements for Integrity Property*

## 6.9 The Availability Security Attribute

*Availability requires that a resource is usable during a given time period, despite attacks or failures.* To verify availability, one must examine the net effects on the control data related to availability. To avoid having to consider temporal properties, one can *specify non-availability rather than availability* (i.e., specify under what conditions the program's behavior catalog do not apply). The analysis of non-availability would proceed along the same lines as the other security attributes just discussed.

# 7 A Miniature Illustration of CSA Using an FX System

The following notional example illustrates application of the CSA approach to determine whether a security requirement is satisfied. Suppose that the specification of the requirements for one security attribute includes a constraint that the value of the second argument to each call to function XXX must not be equal to 4. A constraint such as this, expressed in terms of concrete data operations and values, could be part of the requirements specification for any of the security attributes previously discussed.

Consider the screen image of a behavior catalog generated by the FX/MC prototype shown in Figure 17. Suppose this is the behavior catalog for a fragment of code that executes immediately before the program calls function XXX with the second argument equal to the value stored in register EAX. The computed behavior highlighted as section A in Figure 17 is the behavior with respect to register values for the condition: *?(1 & EAX) == 0*). This condition means "if the value of register EAX at the beginning of this fragment of code is even." As shown on the figure, if this condition is true, the value of register EAX after executing this fragment of code is equal to the initial value of register ECX, and therefore the value of the second argument to function XXX will be equal to the initial value of register ECX. In contrast, the computed behavior highlighted as section B of Figure 17 is the behavior with respect to register values for the condition: *?(1 & EAX) != 0*). This condition means "if the initial value of register EAX is odd." As shown in the figure, if this condition is true, the value of register EAX is unchanged and therefore the value of the second argument to function XXX will be equal to the initial value of register EAX. Thus, the security constraint is not satisfied because, under either of these conditions, we can't certify that the value of the second argument will never be equal to 4.

This analysis takes place in seconds, eliminating the need to study and understand the code by manual means.

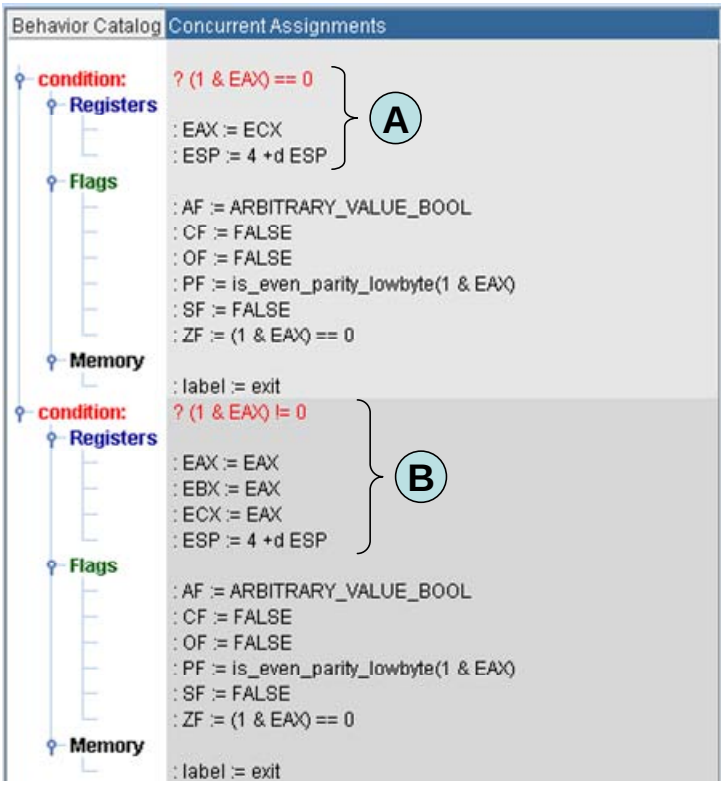


*Figure 17: The Behavior Catalog Computed by the FX/MC Prototype*

# 8 Broader Implications: Improving Security Engineering Practice

Computational security attribute analysis is a step toward a computational security engineering discipline. It can transform security engineering by rigorously defining security attributes of software systems and replacing or augmenting labor-intensive, subjective, human security evaluation. Advantages of the CSA approach include the following:

- A rigorous method is used to specify security attributes in terms of the actual behavior of code and to verify that the code is correct with respect to security attributes.
- The specified security behaviors can provide requirements for a security architecture.
- Traceability capabilities can be defined and verified outside of the automated processes.
- Vulnerabilities can be well understood, making it easier to address evolution of code, environment, use, and users.
- The use of constraints provides a mechanism for explicitly defining all assumptions.

CSA technology addresses the specification of security attributes of systems before they are built, specification and evaluation of security attributes of acquired software, verification of the as-built security attributes of systems, and real-time evaluation of security attributes during system operation.

## 8.1 Advantages of using FX-generated Behavior Catalogs

The behavior catalogs generated by function extraction provide a formal, complete, correct specification of behavior of the code that can be automatically generated. The use of these behavior catalogs provides several advantages for security attribute analysis:

- A behavior catalog lends itself to query. In general, source code and executables do not. Thus, the use of behavior catalogs will facilitate risk assessments and other query-driven analyses of security attributes.
- Because the behavior catalog provides a complete description of behavior, it can be used to answer negative as well as positive questions. This distinguishes CSA from other security analysis techniques such as static analysis and formal methods.
- Because the behavior catalog supports examination of the functional transformations on data and does not require examination of the state space, this approach is more scalable

than formal methods approaches. For example, when formal methods are used to examine, say, the integrity attribute, the entire state space must be computed. Integrity exists if the system can never map outside the state space. In contrast, with CSA and the use of behavior catalogs, only the calculated behavior is of interest. Because only the behavior at the boundary of the system has to be calculated, there is no need to examine the entire internal state space.

- If a property can be expressed in code, FX technology can be used to determine if that property holds within a program.
- FX technology can be used to describe the behavior of a function that combines two behaviors. Thus, FX can be used to give the exact description of the composition of the behaviors.
- Corporate policies and "intentions" can be defined in a behavioral format in advance of the design of the architecture and code. Queries to examine the behavior catalog for the presence or absence of desired properties can be developed in parallel with design of the architecture. If pre- and post-conditions are defined behaviorally, they can be used to evaluate all artifacts (i.e., the behavioral catalogs, not just the code).

## 8.2 Open Questions

Behavioral requirements for concurrency and parallel system issues must be addressed. In addition, because computational security attributes and function extraction are emerging technologies, there has been limited experience in applying the technology to large systems. The computational effort involved in analyzing functions with extensive decompositions and in analyzing large numbers of component compositions to yield a system security specification remains to be studied. However, it is clear that the CSA approach can be more effective than formal methods in addressing state space explosion while yielding complete, correct answers.

## 8.3 Next Steps

The plan is to seek sponsorship to develop prototype automation to support application of CSA technology. This automation will be developed based on a vision of human-computer interaction that would complement and amplify human capabilities for reasoning about software security attributes during development and for real-time evaluation of a system's security attributes during operation. These tools will be constructed in accumulating increments to maximize earned value and minimize risk.

CSA supports a usage-centric evaluation of security attributes that can explicitly consider the objectives and constraints of specific execution environments. Such an approach will support modeling, analysis, and evaluation of the security attribute values of software, as constrained by the policies of specific execution environments. In order for this approach to be widely used, tools are needed to support user input and query of security requirements, including automatic mapping of the model of user-specified acceptable function calls and safe behavior

to the code's behavior catalog. The CERT STAR*Lab FX project is developing tools that will be used to compare behavior catalogs. These FX tools, combined with the CSA approach and proposed CSA tools, will support security analysts in the comparison of security attribute requirements and constraints with behavior catalogs, thus providing a mechanism for automated security attribute analysis.

# References/Bibliography

*URLs are valid as of the publication date of this document.*


**[Ahmed 2003]** Ahmed, Tanvir & Tripathi, Anand R. "Static Verification of Security Requirements in Role Based CSCW Systems," 196-203. *Eighth ACM Symposium on Access Control Models and Technologies (SACMAT 2003*). Como, Italy, June 2003. New York, NY: ACM, 2003. http://www.cs.umn.edu/Ajanta/papers/p342-ahmed.pdf.

**[Albert 02]** Albert, Cecilia & Brownsword, Lisa. In collaboration with Bentley, David; Bono, Thomas; Morris, Edwin & Pruitt, Deborah. *Evolutionary Process for Integrating COTS-Based Systems (EPIC*) (CMU/SEI-2002-TR-005, ADA408653). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. http://www.sei.cmu.edu/publications/documents/02.reports /02tr005.html.

**[Bella 2001]** Bella, G., & Paulson, Lawrence. "Mechanical Proofs about a Non-repudiation Protocol," *Fourteenth Int. Conference Theorem Proving in Higher Order Logics, Lecture Notes In Computer Science, 2152*, Pages: 91-104. Edinburgh, Scotland, Sep. 2001. New York, NY: Springer-Verlag, 2001.

**[Cannady 2001]** Cannady, S. & Stockton, T. "Easing the PAIN: How PKI can reduce the risks associated with e-business transactions" Feb 1, 2001. White Plains, NY: IBM Corporation. http://www-128.ibm.com /developerworks/library/s-pain.html.

**[Caralli 2004]** Caralli, R.A. & Wilson, W.R. "The Challenges of Security Management," Pittsburgh, PA: CERT, Software Engineering Institute, Carnegie Mellon University, 2004. http://www.cert.org/archive/pdf/ESMchallenges.pdf.

**[Chevalier 2004]** Chevalier, Y. & Vigneron, L. "Strategy for Verifying Security Protocols with Unbounded Message Size," *Journal of Automated Software Engineering 11*, 2 (April 2004): 141-166. Norwell, MA: Kluwer Academic Publishers. http://www.avispa-project.org/papers /ChevalierV-JASE04.ps.

**[Collins 2005]** Collins, R.; Walton, G.; Hevner, A.; & Linger, R. *The CERT Function Extraction Experiment: Quantifying FX Impact on*

*Software Comprehension and Verification* (CMU/SEI-2005-TN-047, ADA442865), Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/05.reports/05tn047.html.

**[FDIC 2004]** Federal Deposit Insurance Corporation, U.S. Government. "Legislative and Regulatory Responses to Identity Theft." http://www.fdic.gov/consumers/consumer/idtheftstudy/legislative.html (2004).

**[Hevner 2001]** Hevner, A.; Linger, R.; Sobel, A.; & Walton, G. "Specifying Large-Scale, Adaptive Systems with Flow-Service-Quality (FSQ) Objects," *Proceedings of the 10th OOPSLA Workshop on Behavioral Semantics*, Tampa, FL, Oct. 2001. New York, NY: ACM Press, 2001.

**[Hevner 2002]** Hevner, A.; Linger, R.; Sobel, A.; & Walton, G. "The Flow-Service-Quality Framework: Unified Engineering for Large-Scale, Adaptive Systems," *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. Big Island, Hawaii, January 7-10, 2002. Los Alamitos, CA: IEEE Computer Society Press, 2002.

**[Hevner 2005]** Hevner, A.; Linger, R.; Collins, R.; Pleszkoch, M.; Prowell, S.; & Walton, G., *The Impact of Function Extraction Technology on Next-Generation Software Engineering* (CMU/SEI-2005-TR-015 ADA441893). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/05.reports/05tr015.html.

**[Hussein 2006]** Hussein, Mureed & Seret, Dominique. "A Comparative Study of Security Protocols Validation Tools: HERMES vs. AVISPA," 497-502. *Proceedings of IEEE International Conference on Advanced Communication Technology ICACT'06*, Phoenix Park, Korea, Feb. 2006. New York, NY: IEEE Computer Society, 2006.

**[ISO/IEC 1996]** International Organization for Standardization/International Electrotechnical Commission. *Information Technology - Open Distributed Processing- Reference Model: Architecture, ISO/IEC 10746-3-1996(E)*, 1996. http://webstore.ansi.org/ansidocstore/product.asp?sku=ISO%2FIEC+10746-3%3A1996.

**[Linger 2002]** Linger, R.; Pleszkoch, M.; Walton, G.; & Hevner, A. Flow-Service-Quality (FSQ) Engineering: Foundations for Network System Analysis and Design (CMU-SEI-2002-TN-019 ADA443474), Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. http://www.sei.cmu.edu/publications/documents/02.reports/02tn019.html.

**[Longstaff 2001]** Longstaff, T. "Auditing Legacy Systems for Security and Survivability," 689-692. *Proceedings of the 23rd International Conference on Software Engineering*. Toronto, Ontario, May 2001. Washington DC: IEEE Computer Society, 2001.

**[Longstaff 1997]** Longstaff, T.; Ellis, J.; Hernan, S.; Lipson, H.; McMillan, R.; Hutz-Pesante, L. & Simmel, D. "Security of the Internet," in *The Froehlich/Kent Encyclopedia of Telecommunications vol. 15*, 231-255. New York, NY: Marcel Dekker, 1997. http://www.cert.org/encyc_article/tocencyc.html.

**[NRC 1999]** National Research Council Committee on Information Systems Trustworthiness. *Trust in Cyberspace*. Washington, DC: National Academy Press, 1999. http://www.nap.edu/html/trust.

**[NSA 1987]** National Security Agency, National Security Research Center. *Trusted Network Interpretation, v. 1. 7/31/87. NCSC-TG-005 (Red Book),* 1987. http://www.fas.org/irp/nsa/rainbow/tg005.htm.

**[Pleszkoch 2004**] Pleszkoch M. & Linger, R. "Improving Network System Security with Function Extraction Technology for Automated Calculation of Program Behavior," *Proceedings of 37th Hawaii International Conference on System Sciences.* Big Island, Hawaii, Jan. 2004. Los Alamitos, CA: IEEE Computer Society Press, 2004.

**[Prowell 1999]** Prowell, S.; Trammell, C.; Linger, R.; & Poore, J. *Cleanroom Software Engineering: Technology and Process.* SEI Series in Software Engineering. Reading, MA: Addison Wesley Longman, 1999.

**[Rusinowitch 2003]** Rusinowitch, M. "Automated Analysis of Security Protocols," *Proceedings of the 12th International Workshop on Functional and (Constraint) Logic Programming (WFLP'03),* Valencia, Spain, June 2003. *Electronic Notes in Theoretical Computer Science 86,* 3, 2003. http://www.avispa-project.org /papers /valence2.pdf.

**[Rusinowitch 2003]** Rusinowitch, M. & Turuani, M. "Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete." Theoretical Computer Science 299, 1-3 (April 2003) 451-475. http://www.avispa-project.org/papers /np-complete-tcs03.ps.

**[Santiago 2005]** "Study for Automatically Analyzing Non-repudiation," 151-171. Actes du 1er Colloque sur les Risques et la Sécurité d'Internet et des Systèmes, CRiSIS. Bourges, France, Oct. 2005. http://www.avispa-project.org/papers/SantiagoV-CRiSIS05.pdf.

**[Schneider 1996]** Schneider, S. "Security Properties and CSP" 174. Proceedings of the 1996 IEEE Symposium on Security and Privacy. Oakland, CA: May 1996. Washington, DC, USA: IEEE Computer Society, 1996.

**[Schneider 1998]** Schneider, S. "Verifying Authentication Protocols in CSP," IEEE Transactions on Software Engineering, 24, 9, (Sep. 1998) 741-758.

**[US OCR]** U.S. Government Office for Civil Rights – HIPAA. "Medical Privacy - National Standards to Protect the Privacy of Personal Health Information." http://www.hhs.gov/ocr/hipaa/finalreg.html and http://www.hhs.gov/ocr/AdminSimpRegText.pdf (2006).

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. AGENCY USE ONLY | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| (Leave Blank) | December 2006 | Final |

| 4. TITLE AND SUBTITLE | 5. FUNDING NUMBERS |
|---|---|
| Technology Foundations for Computational Evaluation of Software Security Attributes | FA8721-05-C-0003 |

6. AUTHOR(S)

Gwendolyn H. Walton, Thomas A. Longstaff, Richard C. Linger

| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) | 8. PERFORMING ORGANIZATION REPORT NUMBER |
|---|---|
| Software Engineering Institute<br>Carnegie Mellon University<br>Pittsburgh, PA 15213 | CMU/SEI-2006-TR-021 |

| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER |
|---|---|
| HQ ESC/XPK<br>5 Eglin Street<br>Hanscom AFB, MA 01731-2116 | ESC-TR-2006-021 |

11. SUPPLEMENTARY NOTES

| 12A DISTRIBUTION/AVAILABILITY STATEMENT | 12B DISTRIBUTION CODE |
|---|---|
| Unclassified/Unlimited, DTIC, NTIS | |

13. ABSTRACT (MAXIMUM 200 WORDS)

In the current state of practice, analysis of the security attributes of software systems is typically carried out through subjective evaluations by security experts who accumulate system knowledge in bits and pieces from architectures, specifications, designs, code, and tests. In contrast, this report describes foundations for a new computational security attributes (CSA) technology. This innovative approach provides precise computational methods for defining and analyzing security attributes based solely on the data and transformations of data found within programs. CSA permits security attributes to be evaluated through automatable analysis of the functional behavior of programs. The technology can support specification of security attributes of systems before they are built; specification and evaluation of security attributes of acquired software; verification of the as-built security attributes of systems; and real-time evaluation of security attributes during system operation.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| software security attributes, function extraction technology, behavioral requirements of security attributes, computational evaluation | 51 |

16. PRICE CODE

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UL |